Relazione Progetto
Corso Sistemi Distribuiti

Antonio Cardace, Michele Cucchi, Federico Fossemò
Corso Informatica Magistrale
Università di Bologna

Aprile 2016

1 Abstract

Il progetto implementa la realizzazione in modalità distribuita multinodo, del gioco di carte da tavolo UNO.
La relazione descrive in modo particolare i problemi affrontati, le soluzioni proposte, quelle scartate e l'implementazione definitiva.

2 Introduzione

L'attività di progetto ha visto la realizzazione di un'implementazione software del gioco da tavolo UNO, sviluppandolo in modalità distribuita sfruttando il paradigma *Distributed Object Computing* ed in particolare i costrutti del linguaggio Java **RMI** *Remote Method Invocation*.
Si è cercato di realizzare il lavoro enfatizzando al massimo la natura distribuita, evitando dove possibile di utilizzare paradigmi centralizzati puramente *client-server*. La maggior parte del lavoro è stata concentrata sull'analisi e lo sviluppo di soluzioni per la gestione dei crash ai nodi distribuiti, evitando failure sicuramente bloccanti per l'andamento del gioco.
La relazione è divisa in sotto sezioni:

- *Aspetti progettuali*: descrive la struttura del sistema software realizzato, i problemi affrontati insieme alle soluzioni proposte e scartate
- *Aspetti implementativi* descrive le scelte implementative specifiche dell'architettura realizzata, unitamente ai diagrammi delle classi ed interazioni UML
- *Valutazione* confronta le soluzioni realizzate con lo stato dell'arte
- *Conclusioni* commenti conclusivi e possibili miglioramenti

3 Aspetti progettuali

3.1 Descrizione del gioco

Un giocatore può vincere una partita di UNO quando rimane senza alcuna carta in mano, quindi l'obiettivo del gioco è perdere più carte possibile nel tempo più breve possibile.
All'inizio del gioco il mazziere assegna 7 carte a caso a ciascun giocatore, lasciando le restanti carte in un mazzo sul tavolo a dorso coperto.
La prima carta della pila viene lasciata visibile, perchè sarà la carta iniziale del gioco, formando la prima della pila scarti. I giocatori procedono a turni, in senso orario, a partire da quello alla sinistra del mazziere, lasciando sul tavolo una carta delle proprie sette, che abbia stesso colore o stesso numero di quella lasciata scoperta. c'è la possibilità di utilizzare una carta speciale, ma se sono colorate devono essere compatibili con quella scoperta. Non è possibile scartare più di una carta per turno e nel caso un giocatore non abbia carte giocabili, deve prenderne una dal mazzo, se è giocabile la dovrà spendere immediatamente altrimenti passerà il turno.
Esistono comunque carte speciali che producono vantaggi o svantaggi per il giocatore.

- **Carta Divieto** provoca la perdita del turno di gioco al giocatore successivo, nel caso di due soli giocatori il giocatore che trova la carta può rigiocare immediatamente
- **Carta Inversione** provoca l'inversione del senso dei turni di gioco, fino alla prossima uscita della carta
- **Carta +2** impone al giocatore del turno successivo di prendere 2 carte
- **Carta +4** impone al giocatore successivo di prendere 4 carte, chi gioca la carta può scegliere il prossimo colore
- **Carta Jolly** giocabile in ogni momento, consente a chi l'ha giocata di decidere il colore giocabile dal giocatore successivo

3.2 Architettura del sistema

Il gioco sviluppato rispetta esattamente le regole descritte precedentemente, si svolge a turni e per scelta progettuale prevede un minimo di 4 ed un massimo di 8 giocatori, ognuno dei quali si trova in un singolo host separato.
La presentazione ed il tavolo di gioco sono costruiti graficamente rispecchiando fedelmente forme e simbologie delle carte reali. La costruzione grafica cerca di approssimare il tavolo da gioco reale.
Il sistema è costituito da due tipi di entità software: i **peer** ed il **server**, che interagiscono per avviare ed avanzare il gioco.

3.2.1 Server

Il server è il componente software che esegue in una sola copia e funge da punto di sincronizzazione e coordinamento iniziale per i nodi dei giocatori. Fornisce

delle primitive remote di registrazione per memorizzare i giocatori che intendono partecipare al gioco, ritornando in risposta alla richiesta di registrazione un identificativo numerico univoco. Il server mantiene inoltre, il numero dei giocatori registrati per verificare il realizzarsi della condizione di partenza del gioco, ovvero che il numero dei giocatori in attesa sia compreso tra il numero minimo ed il numero massimo consentiti. Appena viene superato il numero minimo di giocatori, parte un timer, che alla scadenza fa avviare il gioco anche se non è stato raggiunto il numero massimo di giocatori.
Il server esaurisce la sua funzione all'avvio del gioco.

3.2.2 Peer

I peer sono i nodi client dei giocatori, distribuiti su vari e diversi host, implementano l'ambiente grafico, la logica, gli algoritmi di fault tolerance e gestione distribuita del gioco.
I diversi nodi peer eseguono una copia identica del software, varia solo l'identificativo numerico univoco assegnato all'host.

3.2.3 Fasi di gioco

Il sistema di gioco prevede due fasi principali di svolgimento in cui il paradigma di comunicazione tra gli host cambia completamente.
Nella fase di inizializzazione il gioco parte con i nodi dei giocatori attivi in modalità **client-server**, per sfruttare le funzionalità di registrazione e sincronizzazione fornite dal server.
Nella fase di gioco principale il paradigma di comunicazione tra i nodi cambia passando ad una modalità completamente **peer2peer**, in cui ogni host mantiene la visione del gioco per il proprio giocatore e si sincronizza con gli altri per mantenere lo stato consistente, anche nel caso di crash o down di nodi.

3.3 Implementazione distribuita

L'implementazione distribuita del gioco descritto impone l'analisi e la gestione dei problemi che ne derivano, in particolare emergono alcuni problemi specifici:

3.3.1 Registrazione ed identificazione giocatori

Gli host dei giocatori in fase di gioco avviato comunicano tra loro direttamente secondo il paradigma **peer2peer**, ma prima di avviare il gioco è necessaria una sincronizzazione dei giocatori presso un punto comune, anche per ragioni di identificazione reciproca. La sincronizzazione iniziale viene realizzata tramite un'implementazione temporanea, solo per l'avvio, di un sistema **client-server**.
Il server è l'host comune, presso il quale i nuovi giocatori devono registrarsi, per *iscriversi* al gioco. L'operazione di registrazione restituisce anche un identificativo numerico univoco, che consente di riconoscere inequivocabilmente i giocatori.
Al termine delle operazioni di registrazione, una volta soddisfatta la condizione del numero minimo di giocatori, il gioco verrà avviato e di conseguenza gli host si scambieranno informazioni direttamente tra loro, rendendo superflua la presenza del server.
Per la realizzazione sono state pensate due soluzioni:

1. **Server Esterno**: il server è un programma esterno, scritto in un linguaggio diverso da Java che comunica tramite un protocollo comune, ad esempio *HTTP*, senza usare *RMI*, occupandosi del servizio di registrazione e terminando al verificarsi della condizione di avvio del gioco.

2. **Server Integrato**: il server è parzialmente integrato con il software che esegue nei nodi di gioco, è un processo sviluppato nello stesso linguaggio ed utilizza sempre le primitive *RMI*. Il processo server è uno solo e può essere in un host fisico separato dai nodi di gioco. Anche in questo tipo di soluzione, il processo server può terminare nel momento in cui si avvia il gioco.

L'implementazione definitiva realizza la seconda soluzione, scelta per mantenere il più possibile concentrato lo sviluppo, senza disperdersi con tecnologie e linguaggi diversi, ed anche per continuare l'applicazione del paradigma RMI nello sviluppo di una soluzione ad un ulteriore problema.

3.3.2 Gestione dei turni di gioco

L'andamento del gioco procede a turni si rende quindi necessario un paradigma di sincronizzazione tra i vari nodi giocatori, per stabilire inequivocabilmente chi è abilitato al gioco in ogni momento.
La sincronizzazione avviene grazie ad un *token* che viene fatto passare tra gli host in sequenza, realizzando virtualmente *l'anello* dei partecipanti al gioco. La direzione di partenza è stabilita secondo le regole del gioco.
Un giocatore assume il turno quando entra in possesso del token dal nodo che ha appena giocato, al termine del turno spedisce il token al giocatore successivo.
La gestione dei turni risulta molto affidabile ed *autogestita* dagli stessi nodi senza bisogno di entità esterne alla rete peer2peer.

3.3.3 Gestione crash nodi e politica di fault-tolerance

L'implementazione distribuita del gioco realizzata tramite nodi che ne eseguono una copia e comunicano tramite una rete utilizzando una sorta di protocollo peer2peer, è vulnerabile, per la natura stessa del sistema, in caso di perdita della comunicazione tra i nodi o al crash di uno o più di essi.
La perdita di uno più nodi, infatti nel caso specifico del gioco UNO, potrebbe provocare l'impossibilità di proseguire il gioco, ad esempio come nel caso di crash del nodo successivo al termine di un turno oppure dello stesso host attualmente in turno di gioco.
La realizzazione di contromisure in grado di rendere trasparenti al resto dei nodi la perdita di uno o più host, risulta quindi obbligatoria.
Sono state analizzate due possibili soluzioni:

1. **Leader dei nodi**
   Il leader è eletto tra i nodi con *l'algoritmo del bullo*, gestisce il fault tolerance, ricevendo dai singoli host un messaggio di **alive** a cui risponde con un **ACK**, nel caso non riceva il messaggio può considerare il giocatore eliminato facendo proseguire il gioco. Se fosse crashato l'host con il token di turno, il leader ricrea e fa ripartire il token. In caso di crash dello stesso leader, il primo nodo che non riceve l'ACK reindice l'elezione per un nuovo leader.

2. **Token di Fault-Tolerance di Turno e orologi vettoriali**
   Ogni nodo memorizza lo stato del proprio mazzo di carte, lo stato dei mazzi tavolo ed un orologio vettoriale in cui registrare turno e mano di gioco. Queste strutture dati vengono aggiornate ad ogni passaggio di turno, tramite un messaggio broadcast inviato subito dopo la spedizione al nodo successivo del token di turno.
   Nella fase di gioco avviato i nodi si passano un token di *Fault Tolerance* per confermare la *liveness* di ogni host.
   Ogni nodo ha un timeout settato a:

$$(TCPU * nNodi) + \sum Dnodo$$

   dove *TCPU* è la media del tempo di elaborazione dei singoli nodi, *nNodi* è il numero dei nodi, *Dnodo* è il valore medio del ritardo di comunicazione tra 2 singoli nodi.
   L'intervallo temporale descritto da questa formula è settato in un timer presente in ogni nodo che viene resettato ad ogni ricezione del *Fault Tolerance Token*.
   Nel caso di non ricezione del token, il primo timer a scadere genererà l'invio, dall'host in cui è scaduto il contatore, di un messaggio broadcast verso tutti i nodi rimasti, per lanciare la riconfigurazione dell'anello. I nodi risponderanno tutti all'host che ha generato l'evento di riconfigurazione, allegando anche il proprio orologio vettoriale, con cui ricostruire la turnazione di gioco, se si fosse perso anche il token di turno.
   In caso di crash dello stesso nodo generatore della riconfigurazione esiste un ulteriore timeout calcolato in modo da garantire l'impossibilità per due nodi diversi di rilanciare contemporaneamente la riconfigurazione dell'anello.
   La formula di calcolo del tempo di attesa è la seguente:

$$Tattesa = timeout + (timeout * IdPeer)$$

   dove *Tattesa* è il tempo di attesa calcolato, *timeout* è un valore costante configurato e *IdPeer* è l'identificativo univoco del nodo.

L'implementazione realizzata alla fine sviluppa la soluzione numero 2, scelta per evitare il più possibile la centralizzazione di servizi e componenti, mantenendo i nodi paritetici e comunicanti tramite l'astrazione di una rete peer2peer.

4 Aspetti implementativi

5 Valutazione

6 Conclusioni